ORA: The Autonomous Fruit Grocer – From Vision to Cart

Grades 9-12

Teacher Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

This lesson offers an exploration into the world of smart manufacturing and automated logistics. By integrating ORA with Ari as a handheld interface, students move beyond simple coding into the role of systems engineers. This lab demonstrates how digital intelligence, powered by Large Language Models and computer vision, can be harnessed to control physical machinery in a collaborative environment, mirroring the technology used in modern fulfillment centers.

The heart of this challenge is the physical execution of a digital order. Students will program ORA to navigate a 3x3 grid containing various fruit images. Once a fruit is selected by a user via the Ari interface, ORA must precisely move to that specific grid coordinate, "pick" the fruit image, and securely transfer it to a designated grocery basket area. This spatial task requires students to translate visual data from the AI-driven grid into exact ORA coordinates, simulating the sophisticated pick-and-place logic used in global shipping hubs. Successfully moving each image from the grid to the cart requires high-level precision and a firm grasp of coordinate geometry.

Beyond mechanical movement, the lesson emphasizes rigorous logic and safety. Students must implement functional protocols, ensuring fruit isn't selected twice and that the system shuts down correctly, all while adhering to strict safety standards. By mandating low speeds during the development phase and maintaining an accessible emergency stop, students learn the responsibility of working with industrial-grade hardware. This challenge ultimately prepares students for their future work, providing them with a profound understanding of how integrated systems and artificial intelligence converge to solve complex, real-world logistical problems.

If you have not already done so, you should now view the accompanying 14-second time-lapse video that was mentioned in the student guide. This video provides a detailed view of the program at work as ORA moves fruit images from the grid to the grocery cart.

Example Solution to this Challenge

The solution to this challenge provides the opportunity for your students to develop an OzoBlockly Editor program that is modular, dividing the program logic into functions each having a specific purpose. In the solution presented here, variable and function names have been created in a way that promotes self-documentation.

Figure 1 shows a diagram of the grid, located on ORA's base plane. ORA's +x and +y axes are shown in red, with the paper grid entirely in ORA's Quadrant I. Each cell in the grid has a location

number (*LocNumber*) that varies from 0 through 8 as shown. The coordinates x and y shown in red parentheses are the ORA grid coordinates for each of the nine cells. These coordinates may need to be adjusted depending upon where the paper grid is placed on ORA's base plane. The numbers 0-1-2 outside the cells will identify the (*ORAxCell, ORAyCell*) coordinates to be discussed later with Figure 7.

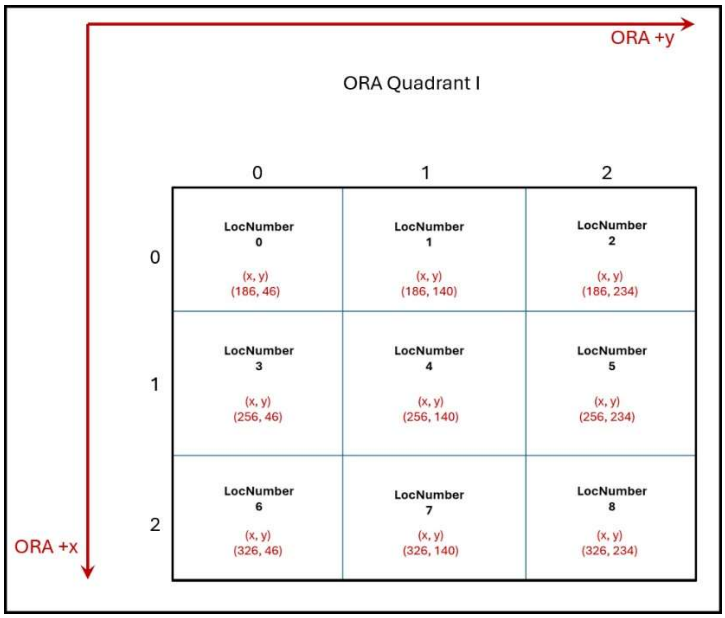

Figure 1

The main program begins with a call to the initialization function shown in Figure 2. Ari's top color is set to white, eliminating the system information that appears automatically. ORA's speed is set to 20%, suggested when doing program development for safety. The vacuum gripper is used by ORA to move the images of the fruit. Arrays used in the program are initialized by a function. To improve documentation throughout the program, a variable called *spaceChar* is set to contain the space character. The number of selected fruit is initialized at zero. Then two functions are called, the first one uses the LLM and camera to get the names of the fruit that have been manually and randomly placed on the paper grid. The second function fills the names of the fruits in the *fruitNames* array.

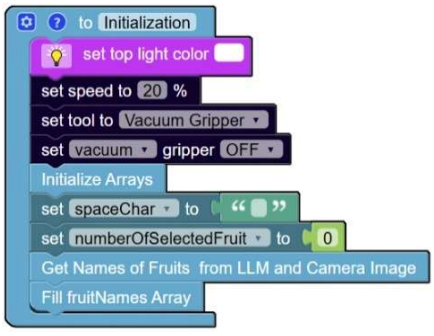

Figure 2

Figure 3 shows the "*Initialize Arrays*" function. The two arrays *ORAxCoordinateIndex* and *ORAyCoordinateIndex* store the six values required to find the *finalORAx* and *finalORAy* values shown in Figure 1. The array *fruitNames* is declared as an array of length nine initially filled with the space character. The array *fruitAlreadySeleted* is defined as an array of nine logic values all initialized as *false*, meaning that initially none of the fruit have been selected. Items in this array will be set to *true* when the corresponding fruit has been manually selected on Ari by the user.

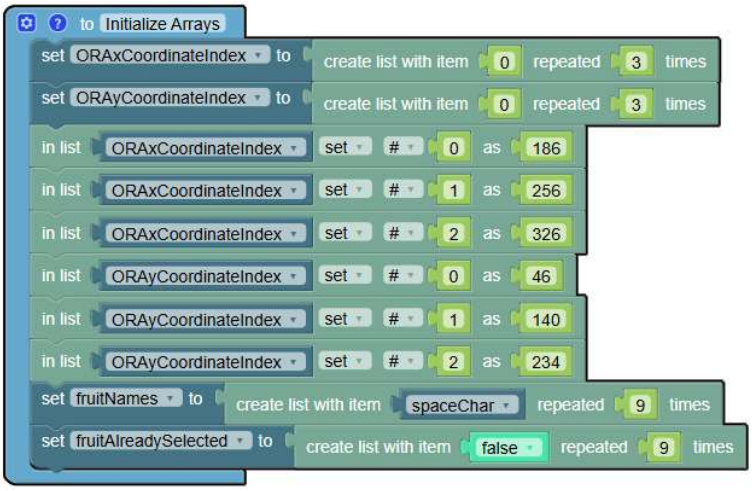

Figure 3

Figure 4 show the "*Get Names of Fruits from LLM Camera Image*" function. The prompt has been designed to return a string with the names of the fruits that the camera sees. The string contains the names of the fruits, each beginning with an upper case letter and with each name separated by a blank space. A typical example for the value of the variable *fruit* might be as follows:

Watermelons Grapes Pineapples Limes Strawberries Apples Bananas Melons Pears

This is printed to the IO console as a debugging aid while developing the prompt.

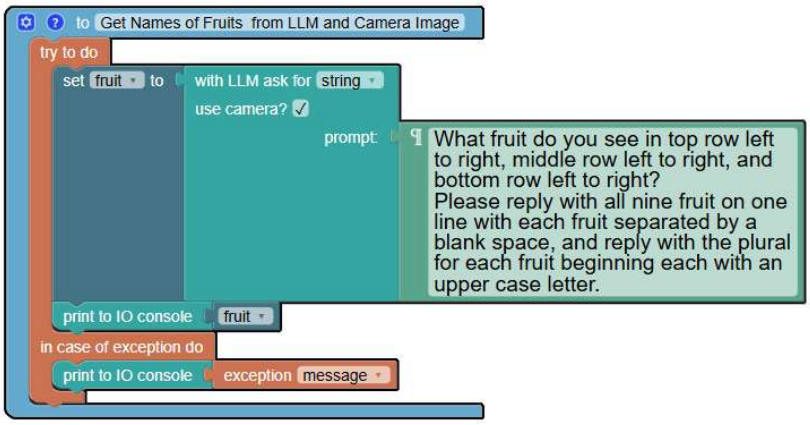

Figure 4

Figure 5 shows the "*Fill fruitNames Array*" function. Filling the *fruitNames* array with the names of the fruit in the string variable *fruit* is a challenging exercise in parsing a string. A loop with *n* varying from 0 to 7 begins by looking for the index of the first occurrence of the space character in the variable *fruit*. The *fruitNames* array then receives the name of the fruit preceeding the space character. The variable *fruit* is then updated so that it contains only the substring following the space character. The fruit name just saved is then printed to the IO colsole for debugging purposes during program development. This iterative process continues through the next to the last fruit. The last fruit, when n = 8, is then handled independently after the loop is completed.

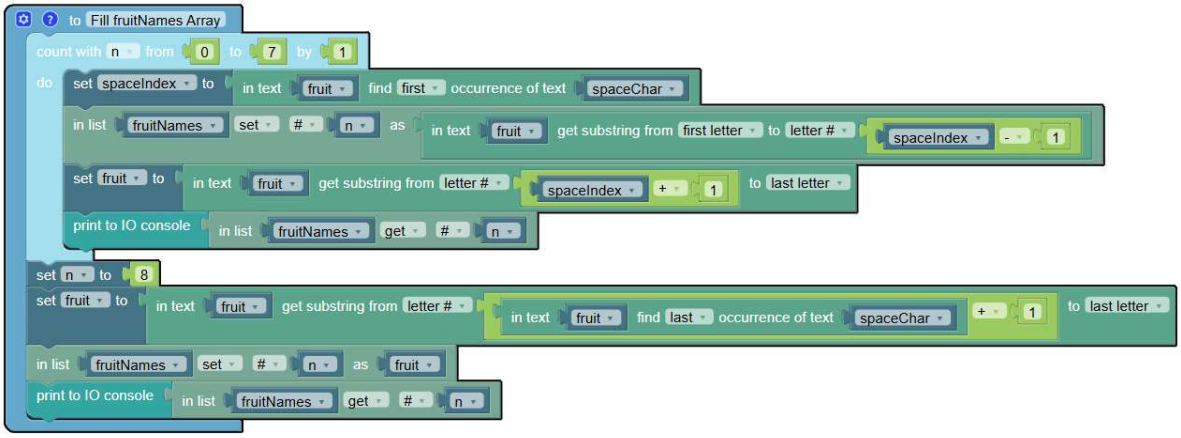

Figure 5

Figure 6 shows the "*Specify the Desired Fruit*" function. This is the function that lists the nine fruit choices on the Ari touch sensitive screen, allowing the user to select a fruit from the list. Note that numbers from 0 through 8, each followed by a blank space, precede each of the fruit names. These numbers are the values of the grid location for each of the fruits.

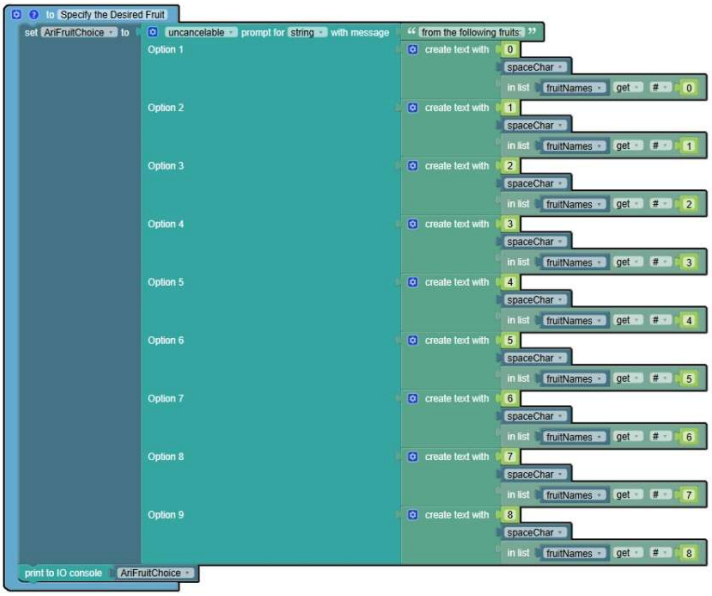

Figure 6

Figure 7 show the "*Set Coordinate Index Based Upon Location Number*" function. The *ORAyCell* coordinate will always be the remainder when *LocNumber* is divided by 3. The ORAxCell coordinate will be 0 for LocNumber 0, 1, and 2. It will be 1 for LocNumber 3, 4 and 5. It will be 2 for LocNumber 6, 7, and 8. As an aid to understanding this, refer to Figure 1.

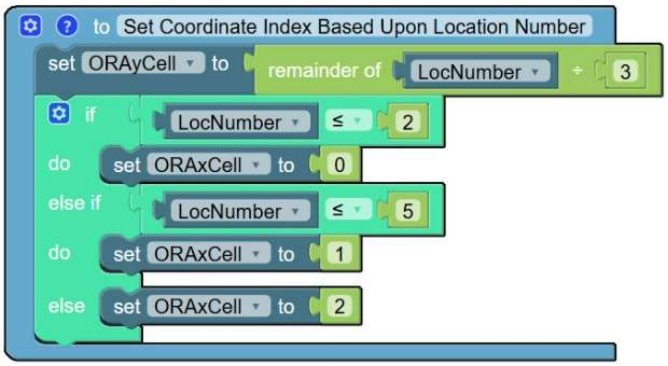

Figure 7

Having now discussed all functions with the exception of the two functions where ORA's arm moves, one where ORA picks up the fruit and the other where ORA drops the fruit into the grocery cart, we are ready to discuss the main program shown in Figure 8.

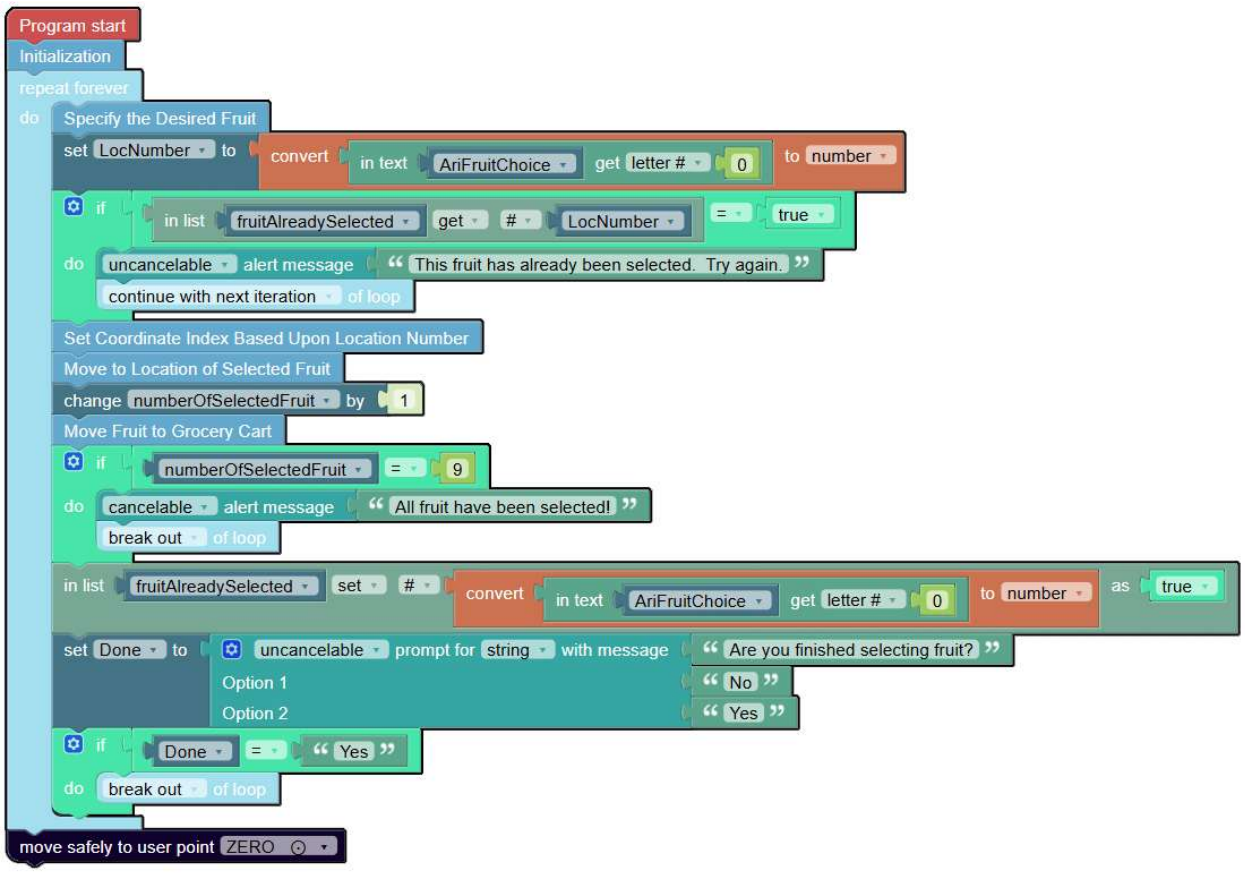

Figure 8

Following the "Initialization" function, nearly all the main program blocks are inside a *repeat forever* loop. There are two situations in which we "break out" of the loop, move ORA to the ZERO position and end the program—when the user has finished selecting fruit and when all fruit have been selected. The *repeat forever* loop does the following:

1. The user selects a fruit using Ari, accomplished by the "*Specify the Desired Fruit*" function.
2. An *if-do* block checks to determine if the desired fruit has already been selected. If so Ari displays a message to that effect. After the user is informed of this, the next iteration of the infinite loop takes place.
3. The function "*Set Coordinate Index Based Upon the Location Number*" is executed and then ORA moves its arm to the desired fruit.
4. The number of selected fruit is increased by 1 and the fruit is moved to the grocery card via the function "*Move Fruit to Grocery Cart*".
5. An *if-do* statement checks to determine if all nine fruit have been selected. If so, Ari informs the user of this fact, and there is a *break out of loop*.
6. The appropriate item in the *fruitAlreadySelected* list is set to the logic value *true*, preventing that fruit from being selected again.
7. Ari asks the user if he/she is finished selecting fruit. If so, there is a *break out of loop*, ORA moves to the ZERO position, and the program terminates.

Figure 9 shows the "*Move to Location of Selected Fruit*" function. This function begins by computing the *finalORAx* and *finalORAy* coordinates from the *ORAxCoordinateIndex* and *ORAyCoordinateIndex* arrays, respectively. The correct indices are in the variables *ORAxCell* and *ORAycell* that were computed in Figure 7. *ORA's* arm is then moved to the final coordinates at a height of 50 mm above the fruit image and then lowered to a height of -3 mm so that the vacuum gripper makes good contact with the image. It is finally raised back to a height of 50 mm.

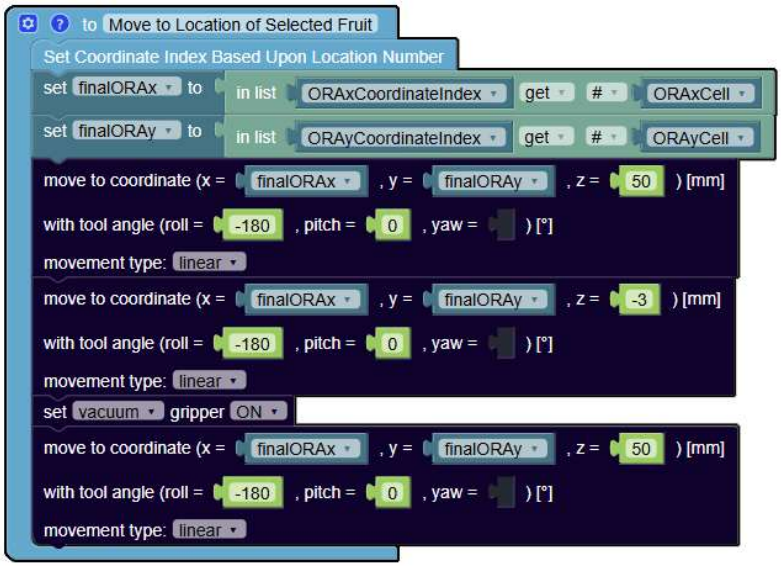

Figure 9

Figure 10 shows the "*Move Fruit to Grocery Cart*" function. The cart is located at ORA coordinates (x, y) = (270, -130). The first move block moves ORA to these coordinates at a height of 50 mm. The second move block lowers the vacuum gripper to a height o 5 mm where the gripper is turned OFF to release the fruit image into the shopping cart. Finally, the gripper is raised once again to 50 mm above the drop off point.

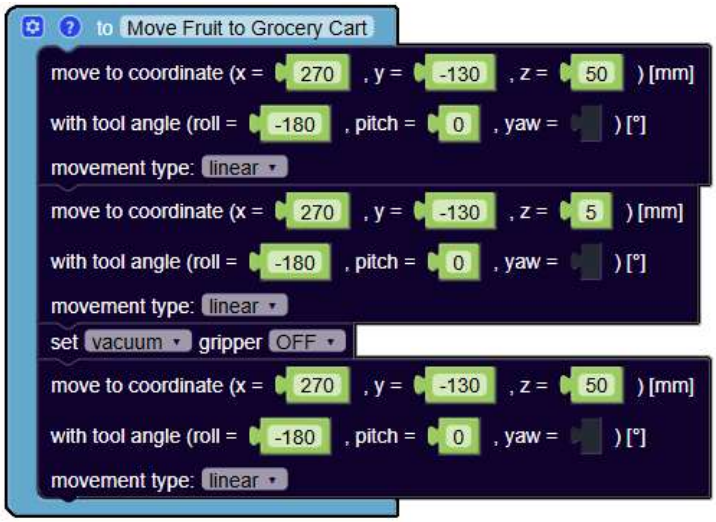

Figure 10

Share Code for this Lesson – pwbthap

Additional Tips

1. The Google mini LLM seems to work very well with this program and doesn't use up LLM credits nearly as fast as the standard model.
2. The camera should be placed so that the paper grid fills most of the camera's field of view, giving better resolution of the image.
3. The paper grid should be well lit to improve the camera's view.

ISTE Standards (International Society for Technology and Education)

Empowered Learner 1.1.d: Students understand the fundamental concepts of technology operations, demonstrate the ability to choose, use and troubleshoot current technologies and are able to transfer their knowledge to explore emerging technologies.

Knowledge Constructor 1.3.d: Students build knowledge by actively exploring real-world issues and problems, developing ideas and theories and pursuing answers and solutions.

Computational Thinker 1.5.c: Students break problems into component parts, extract key information, and develop descriptive models to understand complex systems or facilitate problem-solving.

CSTA (Computer Science Teachers Association) Standards

2-CS-02	Design projects that combine hardware and software components to collect and exchange data.
2-AP-11	Create clearly named variables that represent different data types and perform operations on their values.
2-AP-12	Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.
2-AP-13	Decompose problems and subproblems into parts to facilitate the design, implementation, and review of programs.
2-AP-19	Document programs in order to make them easier to follow, test, and debug.
3A-DA-11	Create interactive data visualizations using software tools to help others better understand real-world phenomena.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3B-CS-02	Illustrate ways computing systems implement logic, input, and output through hardware components.
3B-AP-08	Describe how artificial intelligence drives many software and physical systems.
3B-AP-14	Construct solutions to problems using student-created components, such as procedures, modules and/or objects.